МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский ядерный университет «МИФИ»

ДОМАШНЕЕ ЗАДАНИЕ №1:
«Основы генетических алгоритмов»

По дисциплине «Биоинспирированные алгоритмы решения задач
защиты информации»

Выполнил студент группы Б19-515
Щербакова Александра

Москва, 2023 г.

Задание 1.

Номер по списку группы: 18.

Выполнить программную реализацию простого ГА на одном из языков программирования для поиска максимума функции одной переменной:

$y = cos(2x)/x^2$

$x \in [-20, -2.3]$

Программа простого генетического алгоритма реализована на Python (см. Приложение 1).

Основные параметры алгоритма:

- вероятность мутации = 0.1
- вероятность кроссинговера = 0.8
- число особей в популяции = 50

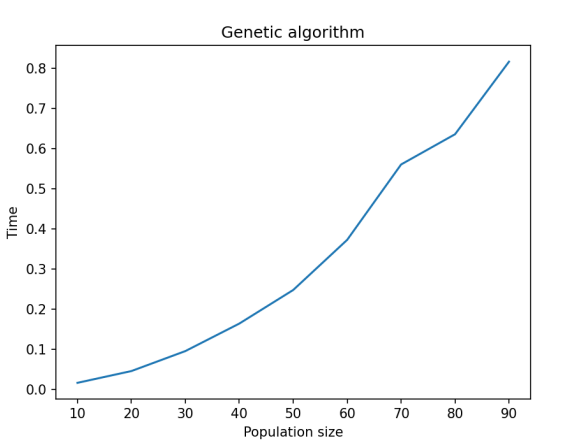
Задание 2.

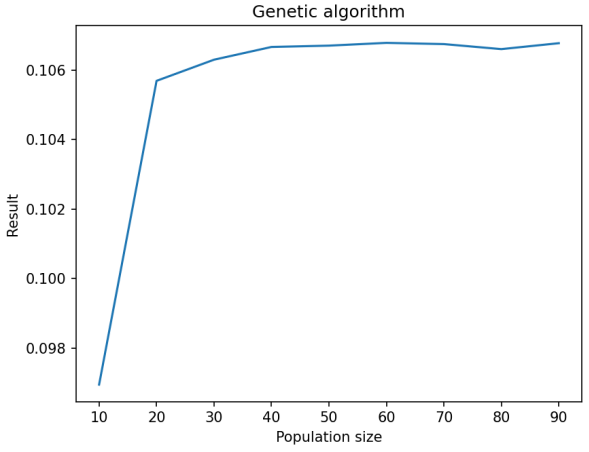
Исследовать зависимость времени поиска, числа поколений (генераций), точности нахождения решения от основных параметров генетического алгоритма:

а) число особей в популяции;

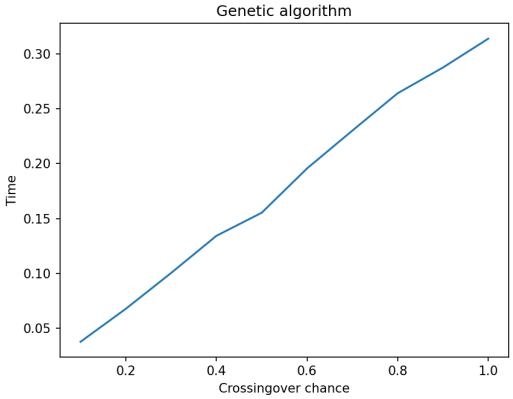
б) вероятность кроссинговера, мутации.

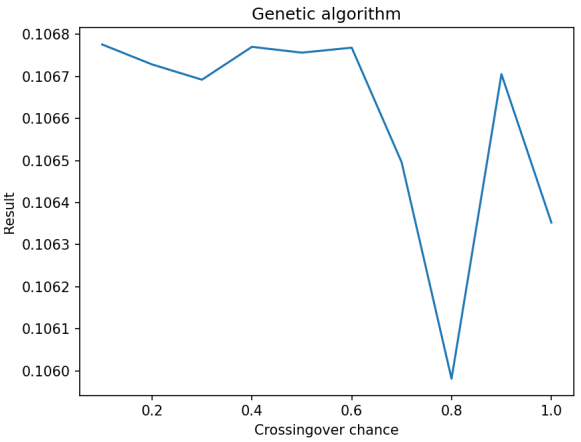
1) Графики зависимости **времени выполнения** алгоритма (слева) и **экстремума функции** (справа) от **количества особей в популяции**:



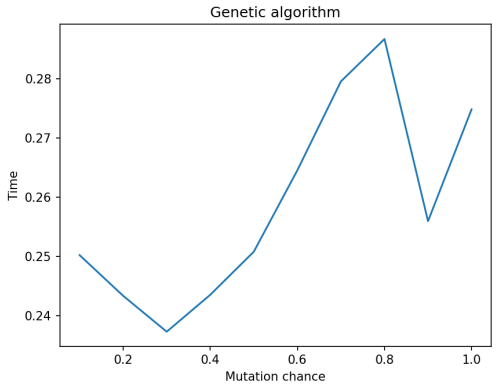


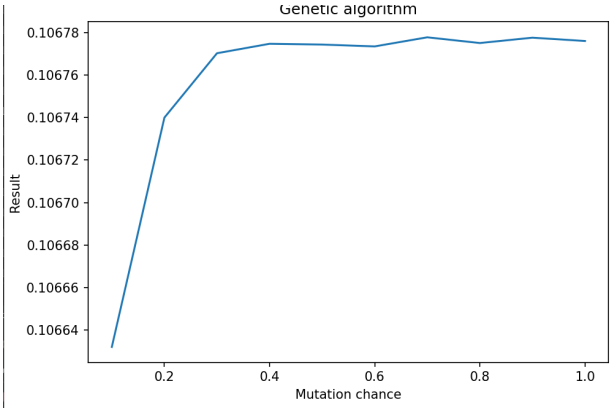
2) Графики зависимости **времени выполнения** алгоритма (слева) и **экстремума функции** (справа) от **вероятности кроссинговера**:





3) Графики зависимости **времени выполнения** алгоритма (слева) и **экстремума функции** (справа) от **вероятности мутации**:

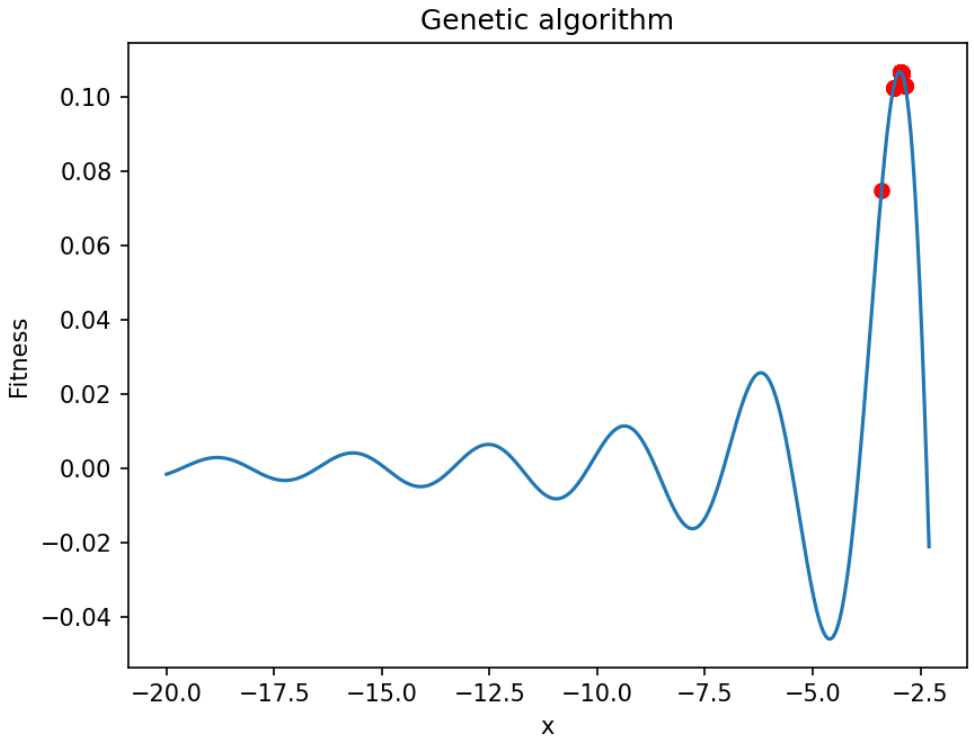




Задание 3.

Вывести на экран график данной функции с указанием найденного экстремума для каждого поколения.

Синяя линия - график исследуемой функции; красные точки - экстремумы для поколений популяции. Точек на графике мало, так как большинство из них слились в одну.



Задание 4.

Сравнить найденное решение с действительным.

При описанных выше параметрах получен следующий результат:

```
Best solution: -2.968893927767727
Fitness: 0.10675126053006807
```

что практически совпадает с реальным результатом:

$$\max\left\{\frac{\cos(2x)}{x^2}\right\} \approx 0.106778 \text{ at } x \approx -2.9797$$

Таким образом, можно сделать вывод о высокой точности работы генетического алгоритма.

Приложение 1.

```
import numpy as np
import matplotlib.pyplot as plt
import time


def fitness_func(x):
  return np.cos(2 * x) / x ** 2


class Gene:
  def __init__(self, x):
    self.x = x
    self.fitness = fitness_func(x)


class GeneticAlgorithm:
  def __init__(self, pop_size, elite_size, mutation_rate, crossover_rate):
    self.pop_size = pop_size
    self.elite_size = elite_size
    self.mutation_rate = mutation_rate
    self.crossover_rate = crossover_rate
    self.population = []
    self.best_gene = None

  def initialize_population(self):
    self.population = [Gene(np.random.uniform(-20, -2.3)) for _ in range(self.pop_size)]

  def evaluate_population(self):
    for gene in self.population:
      if self.best_gene is None or gene.fitness > self.best_gene.fitness:
        self.best_gene = gene

    self.population.sort(key=lambda x: x.fitness, reverse=True)
    elite = self.population[:self.elite_size]
    non_elite = self.population[self.elite_size:]

    # Crossover
    for i in range(len(non_elite)):
      if np.random.rand() < self.crossover_rate:
        partner = np.random.choice(non_elite)
        child_x = (non_elite[i].x + partner.x) / 2
        child = Gene(child_x)
        non_elite[i] = child

    # Mutation
```

```
        for i in range(len(non_elite)):
            if np.random.rand() < self.mutation_rate:
                non_elite[i].x = np.random.uniform(-20, -2.3)
                non_elite[i].fitness = fitness_func(non_elite[i].x)

        self.population = elite + non_elite

    def get_best_gene(self):
        return self.best_gene


# def plot_population(population, func):
#     x = np.linspace(-20, -2.3, 100)
#     y = func(x)
#     plt.plot(x, y, 'k-', label='f(x)')
#     for gene in population:
#         plt.plot(gene.x, func(gene.x), 'ro')
#     plt.xlabel('x')
#     plt.ylabel('f(x)')
#     plt.title('Population')
#     plt.legend()
#     plt.show()


def genetic(pop_size, elite_size, mutation_rate, crossover_rate, num_generations):
    tic = time.perf_counter()
    ga = GeneticAlgorithm(pop_size, elite_size, mutation_rate, crossover_rate)
    ga.initialize_population()

    for i in range(num_generations):
        ga.evaluate_population()
        # best_gene = ga.get_best_gene()

    toc = time.perf_counter()
    return toc - tic, ga.get_best_gene().fitness


def change_crossover():
    time_list_crossover = []
    results_list_crossover = []
    cross_list = np.arange(0.1, 1.1, 0.1)

    # меняем вероятность кроссинговера, графики времени и результата
    for crossover_rate in cross_list:
        t, res = genetic(50, 5, 0.1, crossover_rate, 100)
        time_list_crossover.append(t)
        results_list_crossover.append(res)
```

```
    # график времени от параметра
    plt.plot(cross_list, time_list_crossover)
    plt.title("Genetic algorithm")
    plt.xlabel("Crossingover chance")
    plt.ylabel("Time")
    plt.show()

    # график результата от параметра
    plt.plot(cross_list, results_list_crossover)
    plt.title("Genetic algorithm")
    plt.xlabel("Crossingover chance")
    plt.ylabel("Result")
    plt.show()


def change_mutation():
    time_list = []
    results_list = []
    mutation_list = np.arange(0.1, 1.1, 0.1)

    # меняем вероятность мутации, графики времени и результата
    for mutation_rate in mutation_list:
        t, res = genetic(50, 5, mutation_rate, 0.8, 100)
        time_list.append(t)
        results_list.append(res)

    # график времени от параметра
    plt.plot(mutation_list, time_list)
    plt.title("Genetic algorithm")
    plt.xlabel("Mutation chance")
    plt.ylabel("Time")
    plt.show()

    # график результата от параметра
    plt.plot(mutation_list, results_list)
    plt.title("Genetic algorithm")
    plt.xlabel("Mutation chance")
    plt.ylabel("Result")
    plt.show()


def change_pop_size():
    time_list = []
    results_list = []
    pop_list = range(10, 100, 10)

    # меняем вероятность мутации, графики времени и результата
    for pop in pop_list:
```

```
        t, res = genetic(pop, 5, 0.1, 0.8, 100)
        time_list.append(t)
        results_list.append(res)

    # график времени от параметра
    plt.plot(pop_list, time_list)
    plt.title("Genetic algorithm")
    plt.xlabel("Population size")
    plt.ylabel("Time")
    plt.show()

    # график результата от параметра
    plt.plot(pop_list, results_list)
    plt.title("Genetic algorithm")
    plt.xlabel("Population size")
    plt.ylabel("Result")
    plt.show()



if __name__ == "__main__":
    # start parameters
    # pop_size = 50
    # elite_size = 5
    # mutation_rate = 0.1
    # crossover_rate = 0.8
    # num_generations = 100

    change_crossover()
    change_mutation()
    change_pop_size()


        # график функции + точки лучших генов в популяции
        # x_values = np.linspace(-20, -2.3, 1000)
        # y_values = fitness_func(x_values)
        # plt.plot(x_values, y_values)
        # plt.scatter(x_points, y_points, color='red')
        # plt.title("Genetic algorithm")
        # plt.xlabel("x")
        # plt.ylabel("Fitness")
        # plt.show()
```